Termwork - 3

Title of the Experiment: Implement the Distance Vector Routing algorithm

Objectives:

To implement Distance Vector Routing algorithm to find suitable path for transmission.

Brief Theory:

* When a packet goes from source to destination it will probably pass through many routers until it reaches the destination.
* A router checks a routing table when a packet is ready to be forwarded.
* The table can be either static or dynamic
* Routing protocol is a combination of rules & procedures that let router in the internet inform one another & have been created in response to demand for dynamic routing table.

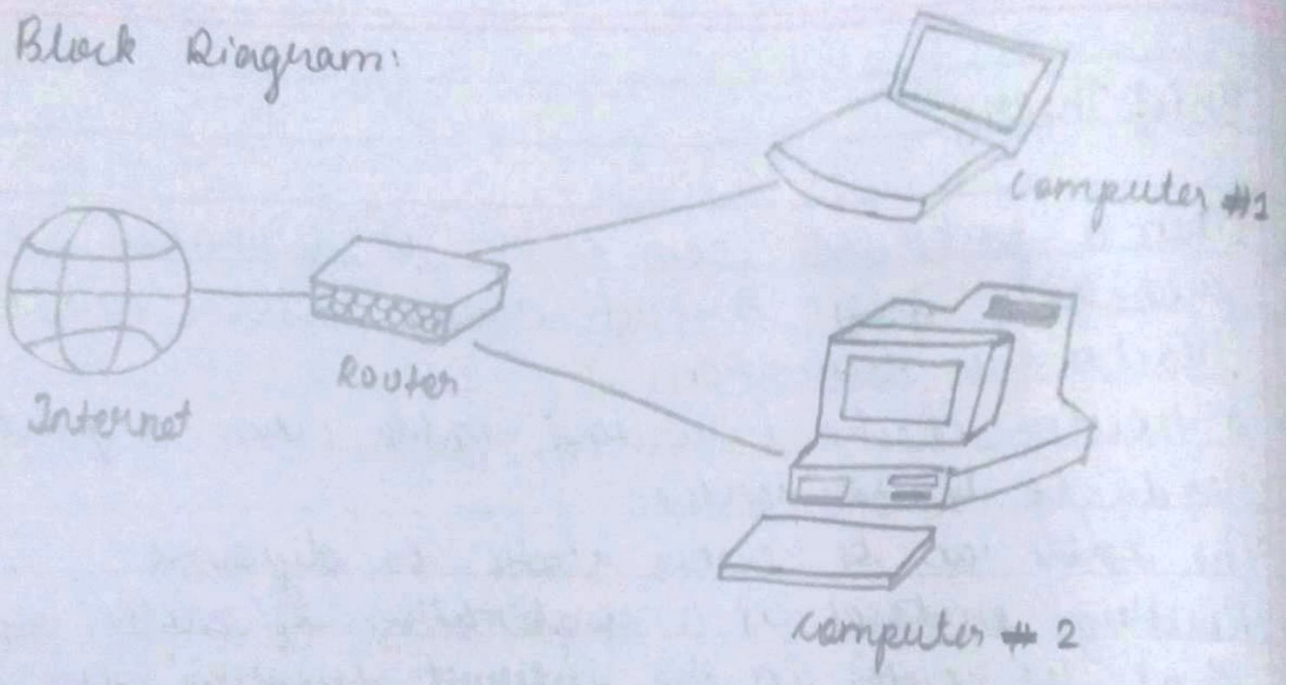
Distance vector Routing:

* Sharing knowledge about the entire autonomous systems
* Sharing only with neighbours
* Sharing at regular intervals.

It is an adaptive / Dynamic algorithm

* Each router maintains a table called vector
* Vector contains no. of hops & delays
* Each router knows best distance to reach another router.
* Also known as Bellmein Ford algorithm
* Each entry has two parts:
  1) Preferred outgoing line
  2) Estimated distance.

Block Diagram:



Algorithm:

1. Send my routing table to all my neighbours whenever my link table changes.
2. When I get a routing table from a neighbour on port P with link metric M:
   - add L to each of the neighbour's table
   - for each entry (D, P', M') in the updated neighbour's table.

   i) if I do not have an entry for D, add (D, P, M') to my routing table

   ii) if I have an entry for D with metric M", add (D, P, M') to my routing table if M' < M"
3. If my routing table has changed, send all the new entries to all my neighbours.

Source Code:

```
#include <stdio.h>
#define nul 1000
#define nodes 10
int no;
struct node {
    int a[nodes][4];
} router[nodes];
void init(int r){
    int i;
    for (i=1; i<=no; i++){
        router[r].a[i][1] = i;
        router[r].a[i][2] = 999;
        router[r].a[i][3] = nul;
    }
    router[r].a[r][2] = 0;
    router[r].a[r][3] = r;
}
void inp(int r){
    int ii;
    printf("\n Enter dist from the node %d to other nodes", r);
    printf("\n Pls enter 999 if there is no direct route\n", r);
    for (i=1; i<=no; i++){
        if (i!=r){
            printf("\n Enter dist to the node %d: ", i);
```

```
        scanf("%d", &router[r].a[i][2]);
        router[r].a[i][3] = i;
    }
  }
}
void display(int r){
  int i,j;
  printf("\n\n The routing table for node %d is as follows:", r);
  for(i=1; i<=no; i++){
    if(router[r].a[i][2] >= 999)
      printf("\n\t\t\t%d \t no link \t no hop", router[r].
                                        a[i][1]);

    else
      printf("\n\t \t \t %d \t %d \t \t %d", router[r].a[i][1],
          router[r].a[i][2], router[r].a[i][3]);

  }
}
void dv_algo(int r){
  int i, j, z;
  for(i=1; i<=no; i++){
    if(router[r].a[i][2] != 999 && router[r].a[i][2] != 0){
      for(j=1; j<=no; j++){
        z = router[r].a[i][2] + router[i].a[j][2];
        if(router[r].a[j][2] > z){
          router[r].a[j][2] = z;
          router[r].a[j][3] = i;
```

```
            }
        }
    }
}
int main(){
    int i,j,a,y;
    char choice='y';
    printf("Enter the number of nodes:");
    scanf("%d",&no);
    for(i=1;i<=no;i++){
        init(i);
        inp(i);
    }
    printf("\n The configuration of the nodes after initialization
                        is as follows:");
    for(i=1;i<=no;i++)
    display(i);
    for(i=1;i<=no;i++)
        dv_algo(i);
        printf("\n The configuration of the nodes after compu-
                    tation of path is as follows:");
        for(i=1;i<=no;i++)
        display(i);
        while(choice!='n'){
          printf("\n Enter the nodes btn which shortest path is to be found:\n");
```

```
        scanf("%d%d",&x,&y);
        printf("\n The length of the shortest path is %d",router[x].
                                                    a[y][z]);
        printf("\n\n continue? (y/n):");
        scanf("%s",&choice);
    }
}
```

Conclusion:

We succesfully implemented Distance Vector Routing algorithm to find suitable path for transmission.

Learning Outcomes:

At the end of the session students will be able to:

→ Understand the concept of Distance Vector Routing

→ Implement 'C' program to find the shortest path between the two nodes

References:

1. Behrouz Forouzan - Data Communications & Networking, Mc Graw Hill Edition
2. Anany Levitin - Introduction to the design & analysis of algorithms.

TERMWORK 3

NAME: SAKSHI B
USN:2GI19CS128
DATE : 10/11/22

OUTPUT

```
Enter the number of nodes:5

Enter dist from the nodes 1 to other nodes
Pls enter  999 if there is no direct route

Enter dist to the nodes 2:2

Enter dist to the nodes 3:999

Enter dist to the nodes 4:999

Enter dist to the nodes 5:999

Enter dist from the nodes 2 to other nodes
Pls enter  999 if there is no direct route

Enter dist to the nodes 1:2

Enter dist to the nodes 3:3

Enter dist to the nodes 4:999

Enter dist to the nodes 5:2

Enter dist from the nodes 3 to other nodes
Pls enter  999 if there is no direct route

Enter dist to the nodes 1:999

Enter dist to the nodes 2:3

Enter dist to the nodes 4:4

Enter dist to the nodes 5:999

Enter dist from the nodes 4 to other nodes
Pls enter  999 if there is no direct route

Enter dist to the nodes 1:999

Enter dist to the nodes 2:999
```

Enter dist to the nodes 3:4

Enter dist to the nodes 5:3

Enter dist from the nodes 5 to other nodes
Pls enter 999 if there is no direct route

Enter dist to the nodes 1:999
Enter dist to the nodes 2:2

Enter dist to the nodes 3:999

Enter dist to the nodes 4:3

The configuration of the nodes after initialization is as follows:

The routing table for node 1 is as follows:

1	0	
2	2	
3	no link	no hop
4	no link	no hop
5	no link	no hop

The routing table for node 2 is as follows:

1	2	
2	0	
3	3	
4	no link	no hop
5	2	

The routing table for node 3 is as follows:

1	no link	no hop
2	3	
3	0	
4	4	
5	no link	no hop

The routing table for node 4 is as follows:

1	no link	no hop
2	no link	no hop
3	4	
4	0	
5	3	

The routing table for node 5 is as follows:

1	no link	no hop
2	2	
3	no link	no hop
4	3	
5	0	

The configuration of the nodes after computation of path is as follows:

The rourting table for node 1 is as follows:

1	0	1
2	2	2
3	5	2
4	7	5
5	4	2

The rourting table for node 2 is as follows:

1	2	1
2	0	2
3	3	3
4	5	5
5	2	5

The rourting table for node 3 is as follows:

1	5	2
2	3	2
3	0	3
4	4	4
5	5	2

The rourting table for node 4 is as follows:

1	9	3
2	5	5
3	4	3
4	0	4
5	3	5

The rourting table for node 5 is as follows:

1	4	2
2	2	2
3	5	2
4	3	4
5	0	5

Enter the nodes btn which shortest path is to be found:1 4

The length of the shortest path is 7

10/11/20

Termwork-4

Title of the Experiment: Using WIRESHARK observe the data transfered in client server communication using UDP & identify the UDP datagram

Objectives:

- To implement WIRESHARK & observe the data transfer between client-server
- To identify the UDP datagram & communication using UDP

Brief Theory:

Wireshark is a software tool used to monitor the network traffic through a network interface. Most widely used network monitoring tool.

Why Wireshark is so popular?

- It has a great GUI as well as conventional CLI
- It is open-source with large community of developers.
- It is free to use.

Wireshark was started with the intention of developing a tool for closely analysing network packets. It was started by Gerald Combs in 1997. Its initial name was Ethereal. It was initially released in July 1998 as version 0.0

Basic Features of Wireshark:

- Packet Monitor
- The packets are shown with the following information
  1. Source address
  2. Destination address
  3. Packet type
  4. Hex dump of the packet
  5. Contents of the packet in text
  6. Source port
  7. Destination port.

What is UDP?

User datagram protocol is another famous transport layer protocol than TCP.

The basic reason is, UDP is a connection less protocol unlike TCP. So, in conclusion when you can compromise some % in reliability but really wanted more speed, UDP is the transport layer protocol you should take.

UDP header is very simple & only 8 bytes
- Source port
- Destination port
- Length
- checksum

UDP applications
- DNS, DHCP, BOOTP, TFTP, RIP etc
- Real time protocol which can't tolerate delay
- Used in some multicasting

Packet analysis:
Lets send some UDP data using Iperf network tool.
Here are the steps:
Step 1: Start Wireshark
Step 2: Run Iperf UDP server at 192.168.1.5 system
Step 3: Run Iperf UDP client at 192.168.1.6 system
Step 4: Stop Wireshark
Step 5: Analysis of captured packets

Source Code:

```
// Client side implementation of UDP client-server model
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#define PORT 8080
#define MAXLINE 1024

// Receiver code
int main(void){
    int sockfd, len, n;
    char buffer[MAXLINE], msg[MAXLINE]= "exit";
    struct sockaddr_in servaddr;

    // Creating socket file descriptor
    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0))<0){
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    printf("[+]Client socket is created.\n");
```

```
memset(&servaddr, 0, sizeof(servaddr));

// Filling server information
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = INADDR_ANY;

while(1){
        printf("Enter message to send to server:");
        fgets(buffer, sizeof(buffer), stdin);
        n = strlen(buffer);
        buffer[n-1] = '\0';
        sendto(sockfd, buffer, sizeof(buffer), MSG_CONFIRM,
            (const struct sockaddr *)&servaddr,
                sizeof(servaddr));
        printf("Message sent.\n");
        if(strcmp(buffer, msg)==0){
            close(sockfd);
            printf("[-]closing server.\n");
            exit(1);
        }else
        recvfrom(sockfd, (char*)buffer, MAXLINE, MSG_WAITALL,
            (struct sockaddr*)&servaddr, &len);
        printf("Server:%s\n", buffer);
}
close(sockfd);
return 0;
}
```

```
// Server side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080
#define MAXLINE 1024

// Driver code
int main(void){
    int sockfd, len, n;
    char buffer[MAXLINE], msg[MAXLINE] = "exit";
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0))<0){
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    printf("[+]Server socket is created.\n");
```

```
memset(&servaddr, 0, sizeof(servaddr));
memset(&cliaddr, 0, sizeof(cliaddr));

// Filling server information
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

// Bind the socket with the server address
if (bind(sockfd, (const struct sockaddr *)&servaddr,
        sizeof(servaddr))<0){
    perror("bind failed");
    exit(EXIT_FAILURE);
}
printf("[+] Bind to port %d\n", 8080);

len = sizeof(cliaddr);
    while(1){
        recvfrom(sockfd, (char*)buffer, MAXLINE,
            MSG_WAITALL, (struct sockaddr*)&cliaddr,&len);
        printf("Client: %s\n", buffer);
        printf("Enter the message to reply to Client:");
        fgets(buffer, sizeof(buffer), stdin);
        n = strlen(buffer);
        buffer[n-1] = '\0';
        if(strcmp(buffer,msg)==0){
```

```
            printf("Disconnected from %s:%d\n",
                    inet_ntoa(cliaddr.sin_addr),
                    ntohs(cliaddr.sin_port));
            break;
        } else {
            sendto(sockfd, buffer, sizeof(buffer),
                MSG_CONFIRM, (const struct sockaddr *)
                    &cliaddr, len);
            printf("Message sent\n");
        }
    }
    close(sockfd);
    return 0;
}
```

Conclusion:

We successfully implemented WIRESHARK & observed the data transfer in client-server using UDP & identified the UDP datagram.

Learning Outcomes:

At the end of the session students will be able to:

→ Understand the significance of wireshark tool

→ Understand how to analyze the UDP datagram using Wireshark

References: W. Richard Stevens, Bill Fenner, Andrew M. Rudoff. "UNIX Network Programming" Volume 1, 3rd edition, Pearson Edu

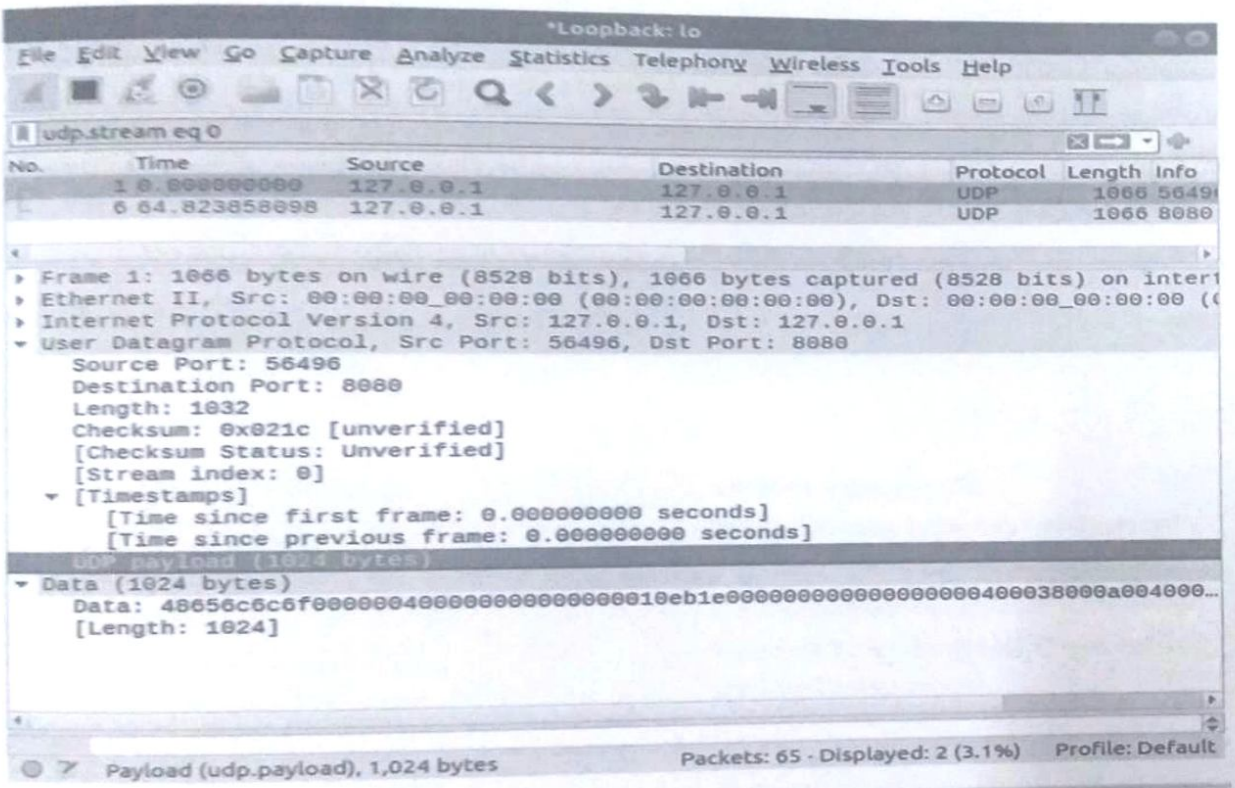
TERMWORK 4

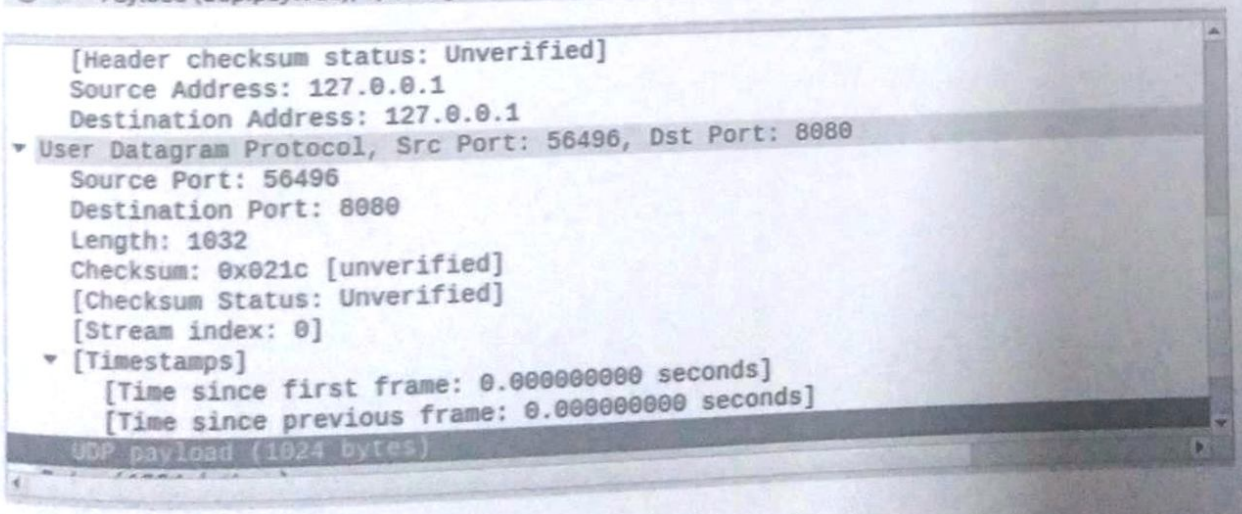
OUTPUT

NAME : SAKSHI BAGEWADI

DATE :24/11/22

USN:2GI19CS128

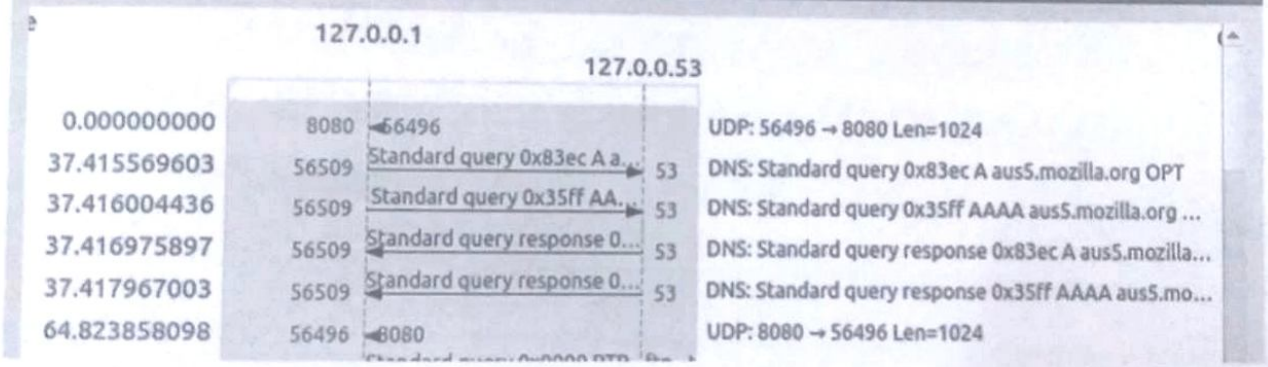




Wireshark · Follow UDP Stream (udp.stream eq 0) · Loopback: lo

```
Hello..@.....@.8.
.@.I.H.....@.....@.....@.....
@.....@.....
.....
.....v.....v>.....v>.....HR.....
.....>.....>.....
.p.....p.....D.....D.....p.....
.....P.td.....v.....v>.....v>.....
.Q.td.....Y.....Y.....
\....d./.....K>./.....@..\.....\.....Z.v/...../....@..
1./....7./...L..\.....\.....\....(o.....T../....@..\.....
\.....
\...p1./.....1./...../.....v/....i...../...
..../....l.u/.....\.....
\.....
\....H../.....)./.....T../....._..g.....4./....@..
3./.....H../.....)./...../.....r|....Hi
Client How are you?.....@.8.
.@.I.H.....@.....@.....@.....
@.....@.....
.....
.....v.....v>.....v>.....HR.....
```

Wireshark · Flow · Loopback: lo



24/4

Termwork-5

Title of the Experiment: Using WIRESHARK analyze three way handshaking connection establishment, data transfer & connection termination in client-server communication using TCP.

Objectives:

- To implement WIRESHARK & analyze three way handshaking connection establishment & data transfer
- To analyze connection termination in client-server communication using TCP.

Brief Theory:

Wireshark is a software tool used to monitor the network traffic through a network interface. Most widely used network monitoring tool.

TCP Analysis using Wireshark:

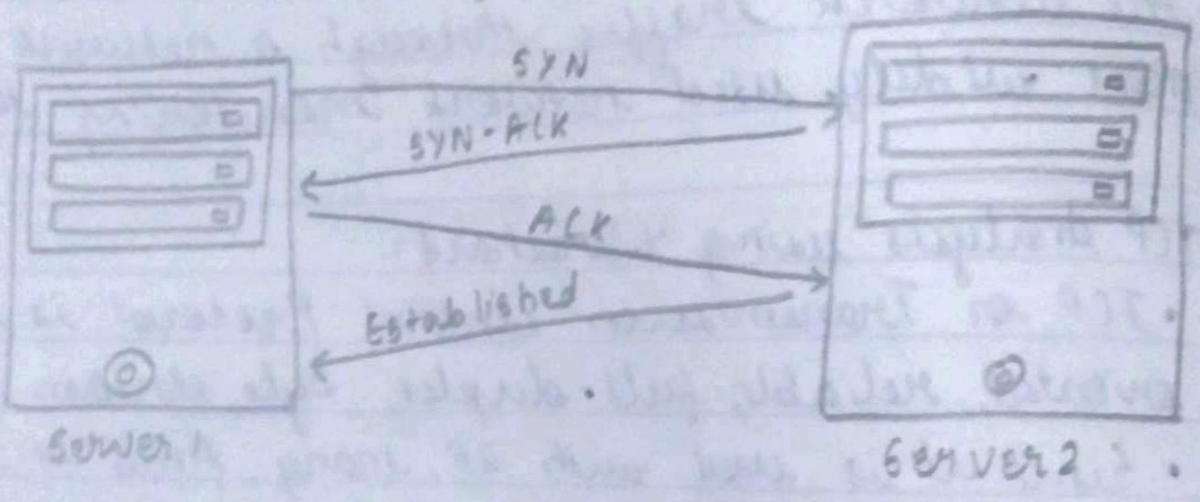
- TCP or Transmission Control Protocol is a connection oriented, reliable, full-duplex, byte stream protocol.
- Since it is used with IP, many times it is also referred to as TCP/IP
- Other features of TCP: sequencing, packet retransmission, acknowledgement.
- From the menu bar, select capture -> options -> interfaces.
- In the interfaces, choose a particular interface, note down its IP, & click the start button of the selected adapter.
- This starts capturing packets
- Run the TCP server & client programs to generate network traffic

Observe the packets ACK, SYN, SYN-ACK is listed on their respective side. Also observe the following details in the packet:

1) Source port
2) Destination port.

Block Diagram:

Three way Handshake / communication



3) TCP segment length
4) Sequence number
5) Next sequence no.
6) Acknowledgement no.
7) Header length

A major section of this TCP packet analysis is the flag section of a packet which gives further in-depth information about the packet. The flag section has the following parameters which are enlisted.

1) Congestion window reduced
2) ECN-Echo
3) Urgent
4) Acknowledgement
5) Push
6) Reset
7) Syn
8) Fin

Further, in the subsections we have:

1) Window size value
2) Checksum
3) Checksum status

The closing side or the local host sends the FIN or finalization packet. The server sends an ACK signaling it has received the FIN packet & sends a FIN packet for confirmation on the closing side. Lastly, the closing side receives the FIN packet & reciprocates by sending the ACK packet thus confirming the connection termination.

Source code:

```
//TCPCLIENT.C
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>

#define PORT 4444
void main (){
  int clientSocket;
  struct sockaddr_in serverAddr;
  char buffer[1024];

 clientSocket= socket(PF_INET, SOCK_STREAM,0);
 printf("[+]Client Socket created Successfully.\n");
 memset(&serverAddr, '\0', sizeof(servAddr));
 serverAddr.sin_family = AF_INET;
 serverAddr.sin_port = htons(PORT);
 serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

 connect(clientSocket,(struct sockaddr*)&serverAddr,
                        sizeof(serverAddr));
```

```
    printf("[+] Connected to Server.\n");
    recv(clientSocket, buffer, 1024, 0);
    printf("[+] Data Recv: %s\n", buffer);
    printf("[+] Closing the connection.\n");
}

// TCPSERVER.C
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 4444
void main(){
    int sockfd;
    struct sockaddr_in serverAddr;

    int newSocket;
    struct sockaddr_in newAddr;
    socklen_t addr_size;
    char buffer[1024];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
printf("[+] Server socket created successfully.\n");
memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(
                                        serverAddr));
printf("[+] Bind to Port no. %d.\n", 4455);

listen(sockfd, 5);
printf("[+] Listening...\n");

newSocket = accept(sockfd, (struct sockaddr *)&newAddr,
                                        &addr_size);

strcpy(buffer, "Hello");
send(newSocket, buffer, strlen(buffer), 0);
printf("[+] Closing the connection.\n");
}
```

Conclusion:

We succesfully implemented wireshark & analyzed three way handshake connection establishment, data transfer & connection termination in client server communication using TCP.

Learning Outcomes:

At the end of the session students will be able to:

→ Understand the significance of Wireshark tool.

→ Understand how to analyze the TCP packets using Wireshark.

References:

W. Richard Stevens, Bill Fenner, Andrew M. Rudoff: "UNIX Network Programming" Volume1, 3rd edition, Pearson 2004.

Termwork 5

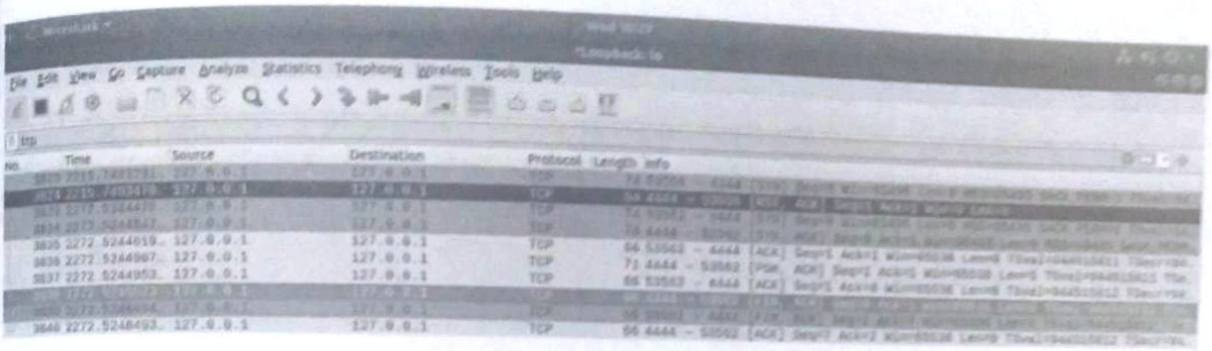
Name: Sakshi Bagewadi

USN: 2GI19CS128

Date: 17/11/22

Output:

TCP Packets



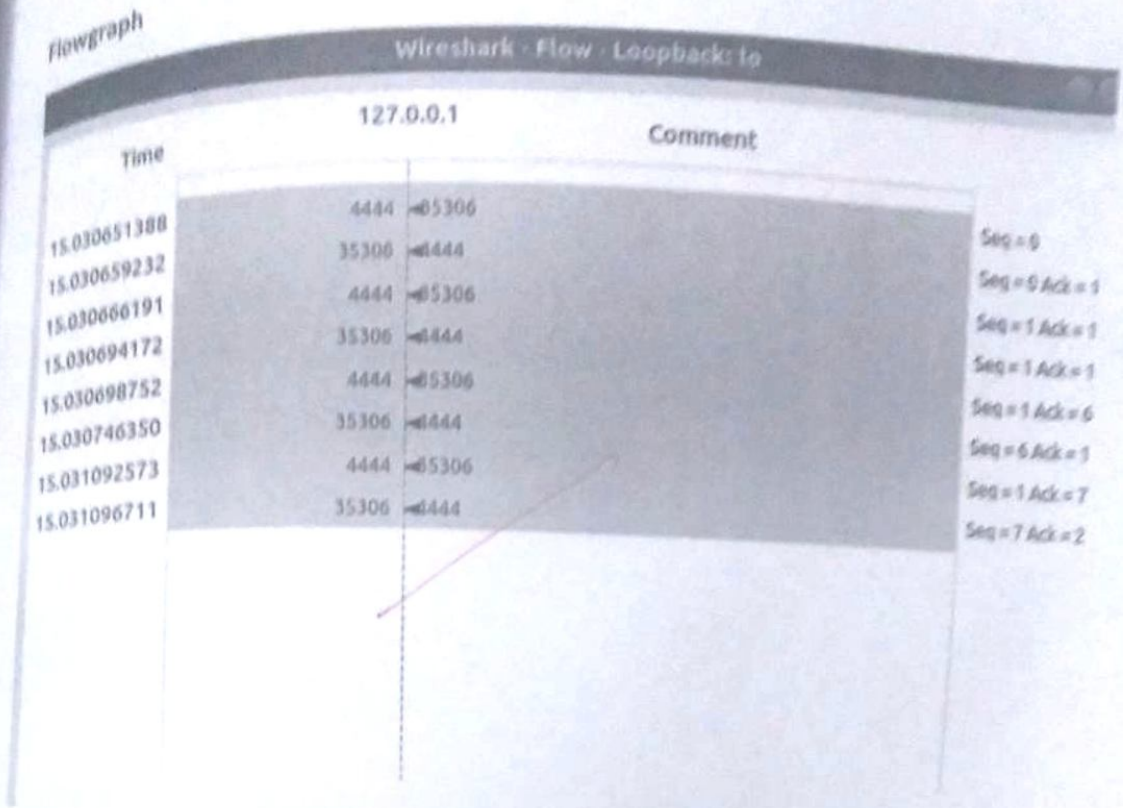
SYN=1 Connection Establishment

```
    Sequence Number (raw): 3582829685
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  ▾ Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    ▸ .... .... ..1. = Syn: Set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ··········S·]
    Window: 65495
    [Calculated window size: 65495]
    Checksum: 0xfe30 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
  ▸ [Timestamps]
```

FIN,ACK=1 Connection Termination

```
  Acknowledgment number (raw): 3582829686
  1000 .... = Header Length: 32 bytes (8)
▾ Flags: 0x011 (FIN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
  ▸ .... .... ...1 = Fin: Set
    [TCP Flags: ·····A···F]
  Window: 512
  [Calculated window size: 65536]
  [Window size scaling factor: 128]
  Checksum: 0xfe28 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
```

Flowgraph



Message Exchanged